REMARKS/ARGUMENTS

Favorable reconsideration of this application, as presently amended and in light of the following discussion, is respectfully requested.

Claims 1-13 and 15-28 are pending in the present application. Claims 1, 2, 16-18 and 22 have been amended, and Claim 14 has been canceled without prejudice.

In the outstanding Office Action, Claim 2 was rejected under 35 U.S.C. § 112; Claims 1-9 and 11-28 were rejected under 35 U.S.C. § 103(a) as unpatentable over Coad et al. (U.S. Patent No. 6,851,107, herein "Coad") in view of Pennel (U.S. Patent No. 6,598,181) and Broughton et al. (U.S. Publication No. 2003/0188299, herein "Broughton"); and Claim 10 was rejected under 35 U.S.C. § 103(a) as unpatentable over Coad in view of Pennel, Broughton and "OMG Unified Modeling Language Specification Version 1.3" (herein "OMG").

Regarding the rejection to Claim 2 under 35 U.S.C. § 112, this claim has been modified in light of the comments noted in the outstanding Office Action. Specifically, "the at least one method call" has been replaced with "the at least one complex method call." Accordingly, it is respectfully requested this rejection be withdrawn.

Claims 1-9 and 11-28 were rejected under 35 U.S.C. § 103(a) as unpatentable over Coad in view of Pennel and Broughton. That rejection is respectfully traversed.

Amended independent Claim 1 is directed to a process for providing a run-time representation of specified characteristics of a previously developed object-oriented software program that includes a plurality of object classes and further includes object related methods belonging to respective object classes. The process includes sensing at least one complex method call by examining the source code of the software program. A plurality of the methods are associated with each of the at least one complex method call. The at least one complex method call includes at least one single method call. The process further includes extracting the at least one single method call from each of the at least one complex method call, the extracting including recursively replacing the at least one single method call with a phase variable, parsing

the phase variable to resolve any polymorphic behavior associated with the extracted at least one single method call, generating a set of information for each of the methods from the at least one single method call, and constructing a representation of interactions between objects of the software program from the information contained in the method information sets. The information set for a particular method contains at least a name of the particular method and the run-time object class of the plurality of object classes to which the particular method belongs. At least one of the object related methods in the software program is a polymorphic method and the run-time object class is determined based on the parsing of the phase variable.

Similarly, independent Claim 16 includes second instructions for parsing a phase variable to resolve any polymorphic behavior associated with an extracted plurality of individual method calls. At least one of the object related single methods in the software program is a polymorphic method and the run-time object class is determined based on the parsing of the phase variable.

Similarly, independent Claim 22 includes means for parsing a phase variable to resolve any polymorphic behavior associated with an extracted plurality of single method calls. At least one of the object related methods in the software program is a polymorphic method and the run-time object class is determined based on the parsing of the phase variable.

In a non-limiting example, Figure 3 illustrates extracting a single method call from a complex method call (see also paragraphs 41-43). As shown in Figure 3 and explained in paragraphs 41-43, complex method calls are isolated and replaced with simple method calls in the form of phase variables. Phase variables are used, *inter alia*, to resolve polymorphism (see, *e.g.*, paragraph 42-44).

Coad relates to a graphical user interface which allows a developer to simultaneously view a graphical and a textual display of source code for the purpose of developing source code (col. 4, lines 38-44; col. 15, lines 17-40). The graphical and textual views are synchronized so that a modification one view is automatically reflect in the other view (col. 2, lines 46-51; col. 4, lines 38-45). Specifically, any modifications to the source code results in a modification of the graphical view which reflects the source code modification (col. 15, lines 41-57). As such, Coad

is intended to be used for source code development rather than for run-time representations and thus does not teach or suggest a ***run-time*** representation of specified characteristics of a previously developed object-oriented software program. Also, because Coad is intended to be used for ongoing source code development, it does not relate to a *previously developed* object-oriented software program. Coad further does not teach or suggest parsing a phase variable to resolve any polymorphic behavior associated with an extracted single method call nor determining a run-time object class based on the parsing of the phase variable. Instead, Coad discloses a "quality assurance (QA) module which monitors the modifications to the source code and calculates the complexity metrics, i.e., the measurement of the program's performance or efficiency, to support quality assurance" (col. 6, lines 18-22). Examples of these metrics are identified in Tables 1-9. Thus, unlike the present invention which parses the phase variable to resolve any polymorphic behavior to determine the run-time object class, the section cited in Coad in the outstanding office with respect to polymorphism is solely for purpose of calculating a complexity metric to support quality assurance (see page 12 of the outstanding office action; see also Coad at col. 9, lines 24-41). Further, because Coad discloses a source code development tool rather a run-time representation, it does not need to handle the complexities associated with identifying the appropriate object class, having a polymorphic behavior, at run-time.

Coad also does not mention phase variables. Applicant respectfully submits that Coad does not even suggest a component of a complex assignment statement with a phase variable. Instead, Coad merely discloses checking for the occurrence of multiple assignments (see Fig. 8B). It does not teach or suggest ***replacing*** these multiple assignments with a phase variable. Applicant further submits that Coad does not teach or suggest recursive replacement.

Pennel relates to a method of debugging multiple function calls by examining object code (see Abstract; col. 2, lines 52-56; col. 5, lines 1-40). Pennel does not teach or parsing a phase variable to resolve any polymorphic behavior associated with an extracted single method call nor determining a run-time object class based on the parsing of the phase variable. In fact, Pennel does not even mention phase variables.

Broughton does not teach or parsing a phase variable to resolve any polymorphic behavior associated with an extracted single method call nor determining a run-time object class based on the parsing of the phase variable. Instead, Broughton discloses that a temporary component is used to simplify binding operations and facilitate storage sharing (paragraph 71). As such, the temporary component disclosed by Broughton is not used for the purpose of resolving polymorphic behavior.

OMG does not overcome the above-noted deficiencies of Coad, Pennel and Broughton.

As stated in M.P.E.P. §2143, a basic requirement for a *prima facie* case of obviousness is that the prior art reference (or references when combined) must teach or suggest all the claim limitations. As the cited references do not teach or suggest the feature of a phase variable to resolve any polymorphic behavior associated with an extracted single method call nor determining a run-time object class based on the parsing of the phase variable, it is respectfully submitted the outstanding Office Action has not created a *prima facie* case of obviousness with regard to independent Claims 1, 16 and 22, and the claims dependent therefrom.

Accordingly, it is respectfully requested this rejection be withdrawn.

Claim 10 was rejected under 35 U.S.C. § 103(a) as unpatentable over Coad in view of Pennel, Broughton and OMG. That rejection is respectfully traversed. No teachings in any of these cited references can overcome the above-noted deficiencies. Accordingly, it is respectfully requested that this rejection be withdrawn for similar reasons as discussed above.

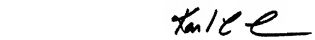
CONCLUSION

In light of the arguments set forth above, Applicants respectfully submit that the Application is now in allowable form. Accordingly, Applicants respectfully request consideration and allowance of the currently pending claims.

It is believed that no additional fees are due at this time. If this is incorrect, Applicants hereby authorize the Commissioner to charge any fees, other than issue fees, that may be required by this paper to Deposit Account No. 07-0153. The Examiner is respectfully requested to call Applicants' Attorney for any reason that would advance the current application to issue. Please reference Attorney Docket No. 124263-1013.

Dated: April 16, 2008

Respectfully submitted,



Karl L. Larson
Registration No. 41,141
ATTORNEY FOR APPLICANTS

GARDERE WYNNE SEWELL LLP
3000 Thanksgiving Tower
1601 Elm Street
Dallas, Texas 75201-4761
(214) 999-4582 – Telephone
(214) 999-3623– Facsimile